By Carl Malamud

Sharing the Wealth: RPCs Help Programs Go Places

Powerful Tools That Help Generate RPCs Can Radically Reduce the Burden When Developing Client and Server Software

Remote procedure calls (RPCs) are a key programming technique for creating distributed applications that transform a network of individual computers into a unified processing entity. With their ability to adapt readily to major network platforms, RPCs let developers write client and server modules that can be moved between computers and operating systems with little modification and recoding.

RPCs can be written entirely by application programmers, an arduous task. Alternatively, they can be generated with the help of RPC development tools such as the aptly named RPC Tools from Netwise (Boulder, Colo.). Such tools make a network appear as one extended local computer, extending a programmer's reach network-wide.

The programming complexities that arise when application software is distributed across different platforms can be an implementer's nightmare. Working with RPC development tools eliminates the need for in-depth knowledge of diverse network protocols and computing platforms, allowing a programmer to concentrate instead on developing the application itself.

RPCs are used to develop support libraries for application program interfaces (APIs) and turnkey applications themselves. In the most general sense, APIs provide software programs with a way to call useful routines that exist on all computers. In the network context, APIs give programs access to remote routines dedicated to a range of activities—such as high-level terminal emulation, distributed database queries, and electronic mail.

The NetWare Core Protocols (NCPs), which give programmers access to NetWare's service routines, is an example of an API without a general RPC-type interface. Calls for NetWare services must be made using a format unique to the NCP API. It does not provide a mechanism for other programs to define and execute their own remote tasks.

For programs to do that in a Novell environment requires Netwise's RPC Tools. Known as NetWare RPC when sold by Novell Inc. (Provo, Utah), the product also has been implemented by several other network vendors as a way of enhancing distributed-application development.

RPCs provide a way for programs in a networked environment to define and execute their own tasks remotely.

RPC Tools furnishes sophisticated communications and control routines that combine with an application's software code to enable servers and clients across a network to exchange data.

Netwise RPC products are not alone in delivering RPC services. The first RPC strategy to gain wide acceptance was designed for Sun Microsystems Inc.'s Network File System (NFS). NFS has been licensed by a broad range of workstation and minicomputer vendors and is widely used in the educational and scientific communities. NFS and the NFS RPC mechanism have been subsumed into Sun's Open Network Computing (ONC) environment.

Another workstation vendor, Apollo Computer Inc. (Chelmsford, Mass.), now a division of Hewlett-Packard Co., developed a powerful RPC mechanism for its Network Computing System (NCS). NCS, like NFS, enjoys a sizable support base. Apollo's RPC recently was accepted as a distributed computing standard by the Open Systems Foundation's (OSF).

Netwise RPC Tools Version 2.0, the current release, is based on ISO Standard 8825 for the encoding of data, known as Abstract Syntax Notation 1 Basic Encoding Rules (ASN.1 BER). Netwise also has committed to supporting the ISO remote operations standards when they mature.

In the interim, Netwise has signed an agreement with Sun to integrate the two vendors' RPC tools. Version 3.0 of Netwise's RPC Tools will support both Netwise and Sun RPC protocols.

While a single RPC standard is unrealistic, some standardization will yield immediate benefits for users. Developers will be able to port programs to a wider variety of computing platforms and network transport mechanisms. Accordingly, networkers will be able to use the strengths of a widening variety of computing platforms.

RPC-developed applications are independent of any particular transport lay-

RPCs Help Programs Go Places

er. The Netwise network library allows the selection of different transport mechanisms at run time. Initially, an application might run in a Novell environment, using IPX and SPX as the network and transport layers. Later, the network could be changed to TCP/IP to allow greater flexibility in choosing computers to put on the network (see Figure 1).

Other user interfaces might be furnished for workstations from Apple Computer Inc., Sun, and Digital Equipment Corp. Additional servers also might be provided. Moving the application into this new networking environment is meant to be easy. Supporting TCP/IP instead of IPX and SPX is a simple matter of picking the appropriate network library. Keeping pace with changing demand for RPC Tools is not difficult; the network library can be readily modified to accept different transport stacks, like the OSI TP protocols.

The Netwise compiler uses the C programming language. Presumably, the application program also is written in C. If the programmer initially had used a standard version of the C compiler, migrating a program would consist principally of recompiling and linking it on the new machine (assuming, of course, that a variety of pitfalls, such as differences in bit order, byte order, and word size) are not encountered.

What makes Netwise's product particularly attractive is that versions of RPC Tools also exist for a variety of other platforms. Among them are all of the 80X86-based microcomputers, as well as machines from DEC, Sun, Hewlett-Packard and many others. A wide range of operating environments are supported as well, including MS-DOS, Microsoft Windows, OS/2, DEC's VMS and Ultrix, and Unix. Network operating systems supported include NetWare, Netbios, TCP/IP, OS/2 LAN Manager, DECnet, and OSI.

RPCs FOR DATABASE ACCESS

A simple database management system (DBMS) is typical of the applications using procedures that interact via interprocess communication. A DBMS program for a single machine might be structured as two procedures: The first manages the user interface and the second handles data files. The DBMS accepts requests through the user interface, which in turn calls upon the file-handling procedure to retrieve the desired results. The results are returned to and presented by the user interface procedure.

When the developer is satisfied that the code will do what is intended, the two procedures are compiled from source code (in C or Fortran, for example) into machine-language object files. Then the procedures, including any support libraries, are linked together into an executable program. Among other things, linking is the process of registering all of the procedures (two in this case) in a table so they can find one another.

When the program runs, any procedure may call any other procedure, passing it predefined arguments. The calling procedure relinquishes control of the computer, and the called procedure performs its task and returns the appropriate data and control to the caller.

Of course, this explanation is an oversimplification. A real-world DBMS usually has a large number of procedures, each performing a very specific task. The task of handling the data file would be managed by several procedures, each specializing in some aspect of retrieving data: security, query optimization, or network access, for example.

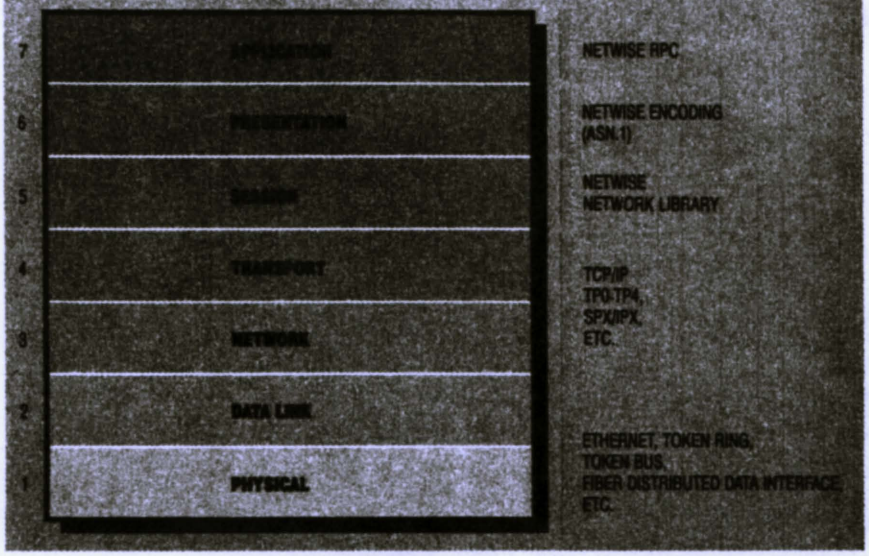
Whether there are two procedures or many, however, the basic mechanism is the same. A procedure on a machine can call another and activate it to perform its task. The caller provides the called procedure with the information it requires (passed arguments) and reads the resulting information (returned values) when the process is complete (see Figure 2).

Distributing the two procedures onto different machines is very attractive. The user interface is a natural candidate for the user's workstation, with its graphics controller and operating system geared to respond to keyboard and mouse input.

The data file–handling procedure, on the other hand, is best suited to run on a dedicated server. A typical network DBMS server, optimized for data access instead of user input, requires a lot of mass storage. A network DBMS server usually is capable of accepting requests from multiple user interface programs, al-

Figure 1: Netwise RPC Tools and the OSI Reference Model

Netwise's RPC Tools provides transport-independent application, presentation, and session layers for developers of distributed applications. RPC Network Libraries, providing session layer support, are available for such transport-layer protocols as Netbios, NetWare SPX, Named Pipes, TCP/IP, DECnet, HP NetIPC, and others.



RPCs Help Programs Go Places

lowing for significant economies of scale.

Several issues complicate the distribution of an application's procedures over networked computers. Among these are locating and forming an association (binding) with the remote procedure; representing the data in a common format; sending and receiving the data; and handling the kinds of errors that do not arise in a single-machine environment.

Resolving these issues for distributed applications is the role of Netwise's RPC Tools. The RPC Compiler transparently converts a series of procedures created to run on a single machine into a series of procedures able to run on several systems. It handles the issues of how the remote procedures are to be activated and how to pass and return complex data structures. It also manages such concerns as the use of pointers and error handling. Further, it provides a syntax for registering, locating, and binding procedures.

FROM ONE TO MANY

In a local environment, the process for writing a program involves writing source code, compiling source code into object code, and linking object code into executable code. In a distributed environment, along with source code, the programmer writes an RPC specification, providing details about where processes should reside, whether they will have permanent or temporary sessions (bindings), and specifications for any customized procedures. Next, the RPC specification and user source code are precompiled by the RPC Tool, producing a mix of user code and RPC-specific code that is then compiled into modules to be run on servers and clients.

When the RPC compiler has finished distributing an application, four additional procedures have been added to the original program: client and server stubs; a dispatcher procedure; pack and unpack procedures; and network and support libraries.

The client and server stubs interface to the calling and called procedures to isolate them from considerations external to their tasks, namely getting the network and other intervening services in their interprocess communication. They provide local emulation of the procedures, which are now remotely located.

The dispatcher procedure makes sure a call gets through and back. The dispatcher calls the pack and unpack procedures to translate data into network-transportable format and remote-host-readable format; returns it to transportable format, and finally back into caller-readable format. The network and support libraries provide the hooks into the selected transport layer via appropriate run-time service and control procedures (see Figure 3).

Since many of the mechanisms in RPC Tools operate by default (a programmer can provide a list of which procedures will be distributed), it is possible for programmers to be totally unaware of the distributed environment—the network is treated as if it were simply one large computer. For complex applications, the programmer may modify each of the service elements in RPC Tools to provide better control or performance.

Since RPCs see networks as one large computer, programmers can be totally unaware of the distributed environment.

Because a network server usually will have more than one remotely callable procedure, RPC Tools provides each server-based program with a special procedure to coordinate the activities of all the callable procedures and their support mechanisms, called a server control procedure (SCP).

SERVER CONTROL PROCEDURES

There are three types of SCPs. The first type handles single-server-to-single-client applications; the second sequentially services requests from multiple clients; and the third, a multitasking version, simultaneously services multiple RPC requests. The third type is designed to run on computers with multitasking operating systems such as OS/2 and Unix.

When programmers define procedures as remote, they must include definitions for any passed arguments and returned values that the procedure will use, as they would for a local procedure declaration. The programmer also can declare that certain variables are accessible globally to several different procedures without having to be explicitly passed.

Global variables are supported in RPC Tools because many existing applications make heavy use of them as part of their interface. Support for global variables makes porting these applications to a distributed environment significantly simpler.

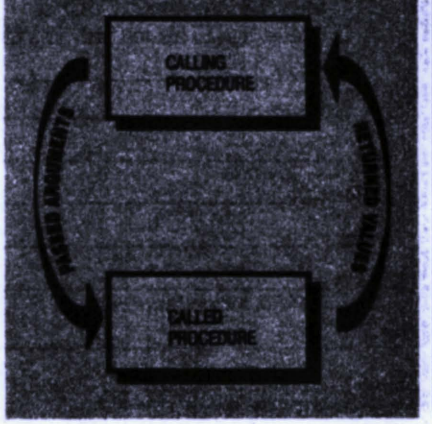
The method of defining data structures for an RPC is the same as that used in the C programming language. Simple data types such as integers or characters can be defined, as can structures. A structure is a combination of several instances of simple data types. Two exceptions to standard C programming techniques exist for the data types of arrays and unions.

The problem with arrays is that often not all of the positions in it are filled for a particular request. An array might be designed to hold 300 rows of data. An incoming request might want only 12. In a one-machine environment, unused space in an array is not a problem (assuming there's enough memory). Over the network, transmitting empty cells can have a dramatic affect on the amount of network traffic produced by any one request.

Netwise handles this difficulty by allowing the programmer to specify a ter-

Figure 2: Keeping It Simple

Remote procedure calls, from the programmer's point of view, are meant to be as simple to write as regular procedure calls, which rely on interprocess communications.



RPCs Help Programs Go Places

mination expression for arrays. After the first unused cell, the programmer may insert the termination value (a null character, for example) into the next cell. The pack procedure will recognize the termination value and stop packing.

The second data structure that requires special handling is the union. A union consists of several elements, only one of which can be active at any one time. Again, in a single-machine environment, the problem with a union is the use of memory. Sending all of the members of a union through the network, when only one is being used, is a deplorable waste of bandwidth.

To supplement the union definition, the RPC procedure specification may include a choice expression. The choice expression is executed at run time to determine which member of the union is active at that time. Only the active member is sent over the network to the remote procedure.

PROCESS BINDING

Binding specifies how the relationship between a remote procedure and the calling program will be established. A binding is formed when two applications have made a connection and are prepared to exchange commands. If the procedure is called rarely, a nonpersistent style of binding is used.

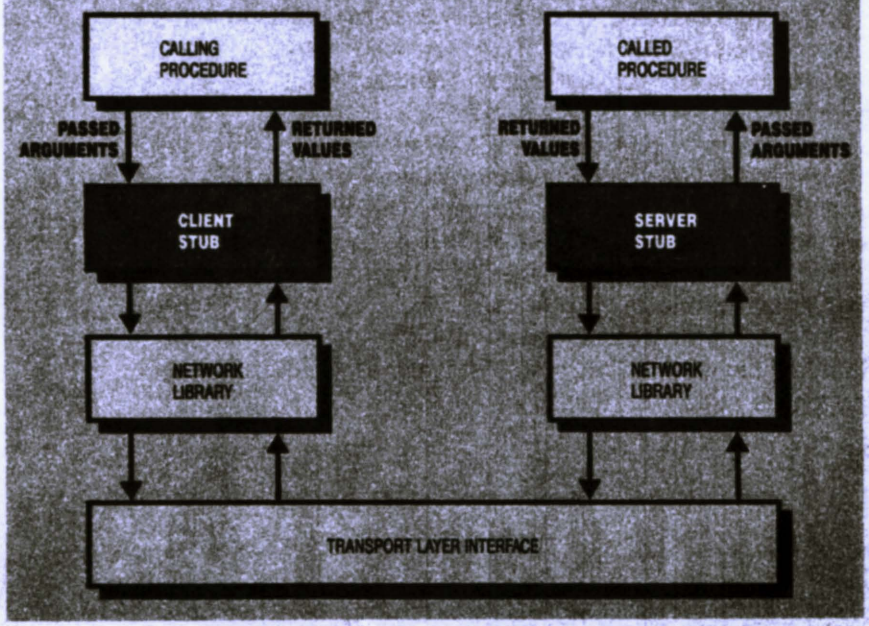
Nonpersistent binding means that a connection is set up on the underlying transport mechanism when the remote procedure is called. As soon as the values are returned, the connection is dismantled. Since connections on a network are a limited resource, the nonpersistent style is used to conserve those resources. The overhead involved in establishing connections makes nonpersistent binding inappropriate for remote procedures that are called frequently by the same caller.

For applications that make many repeated calls to remote procedures, persistent bindings allow several different calls to use the same connection. The connection is set up and then kept open.

For both types of bindings, the address of the remote server can be determined at run time. The address may be specified as a name or, if the remote computer where the server will run is always the same, an internetwork address can be used to specify the target.

Figure 3: Transparent Service Links

Client and server stubs impersonate called and calling procedures to hide the complexity of passing and returning arguments between procedures across the network. The network library provides access to the underlying transport stack for their communications.



It is more and more common for applications to migrate to more powerful hardware because of increased demand for services, so using aliases for services is very important. Using names instead of internet addresses during program development allows the network manager to move a program from one computer to another without making program-level changes on either the client or the server. In a typical scenario, the RPC libraries will take the name of a server or other destination from an application and do a name search of the network that will return an internetwork address.

When the location of an RPC-callable service changes, the initialization procedure in the server control program for the application will inform the name service (i.e., the NetWare bindery, DEC's Domain Name Server) of its new location. Workstations requesting the remote services will be informed by the name service of the current location of the service.

The remote procedure call mechanism is a simple, elegant interface to a distributed computing environment. At the most basic level, it is transparent to the program developer. More sophisticated use is possible both in the RPC mechanism and on the lower levels of the network. Four (or more) RPC mechanisms are now battling for acceptance. DEC, HP/Apollo, and IBM have OSF's Decorum. OSI is developing the Remote Operation (Rose). Sun has the venerable NSF ONC. Finally, there is Netwise, used widely on PC nets and now aligned firmly with Sun's ONC family.

RPCs and similar tools are furnishing the building blocks for distributed applications. Which RPC mechanism will ultimately become the standard? Most likely, there won't be a single standard nor does it matter whether one RPC predominates. In the end, it's not who wins that counts but that the RPCs are here today that can lead to better programs tomorrow. ■

Carl Malamud is a writer and consultant based in San Francisco. This article is excerpted from Malamud's forthcoming *Analyzing Novell Networks*, to be published by Van Nostrand Reinhold.